***ddeedduupp* v0.1**

**User Manual**

April 2, 2016

David T. Ashley (dashley@gmail.com)

Table of Contents

[1 License 3](#_Toc447454215)

[2 Introduction and Overview 3](#_Toc447454216)

[2.1 Brief Description of *ddeedduupp* Functionality 3](#_Toc447454217)

[2.2 Definitions 3](#_Toc447454218)

[2.3 Detailed Description of *ddeedduupp* Functionality 4](#_Toc447454219)

[3 Invocation Example 6](#_Toc447454220)

[4 Technical Description and Internal Operation of *ddeedduupp* 6](#_Toc447454221)

[4.1 Heap Allocation 6](#_Toc447454222)

[4.2 RAM Data Structures 6](#_Toc447454223)

[4.3 Approximate Limits on File Set 6](#_Toc447454224)

[4.4 Approximate Initialization Time 6](#_Toc447454225)

[4.5 Method of Moving Files from *tgt* to *dup* 6](#_Toc447454226)

[4.6 Use on a Changing File Set 7](#_Toc447454227)

[4.7 Use on a Networked Drive 7](#_Toc447454228)

[4.8 Redistributing *ddeedduupp* 7](#_Toc447454229)

[4.9 Rebuilding *ddeedduupp* from Source Code 7](#_Toc447454230)

[4.10 Modifying *ddeedduupp* 7](#_Toc447454231)

[4.11 Redistributing modified *ddeedduupp* 7](#_Toc447454232)

# License

# Introduction and Overview

## Brief Description of *ddeedduupp* Functionality

*ddeedduupp* is a simple *Windows*-only tool for identifying and selectively removing duplicate files within a single directory and its subdirectories on a single disk volume. It was originally devised by the author to eliminate duplicate photos and documents.

*ddeedduupp* has the following strengths and advantages:

* It is free (without cost).
* The full source code is provided with the tool, under a very unrestrictive license. (It is built under *Microsoft Visual Studio*, and can be built under *Microsoft Visual Studio Community Edition*.)
* The program is a single executable that is simply copied onto a computer (no installation, no registry changes, no DLL dependencies, etc.).
* The program will handle large collections of files easily.

*ddeedduupp* has the following weaknesses, limitations, and disadvantages:

* The program is provided for *Microsoft* *Windows* only (it should run on *Windows 7* or newer).
* The program has no notion of files that are *nearly* identical. (As viewed by *ddeedduupp*, files are either identical or not identical.)
* The program is console-based (non-graphical).
* The program does not save context between invocations. (The initialization time for the program may be rather long with large sets of files, as the program begins by calculating the SHA512 cryptographic hash of every file. However, once the program has initialized, operations are quick.)
* The program requires that all files and directories to be analyzed be placed in a single directory on a single disk volume. (For typical home applications, this isn’t a limitation—directories can be moved, analyzed and manipulated using *ddeedduupp*, and then moved back to their original locations.)

## Definitions

Most computer users are familiar with the concepts of file and directory, and that directories may contain files and other directories. However, in describing *ddeedduupp*, it is possible I use nomenclature that is unfamiliar to many users or that is non-standard, so this nomenclature should be defined.

By the *children* of a directory, I mean the files and directories that are contained in a directory (but not the children of any contained directories. Directories may have children, but files may not. Children of a directory may include both files and directories.

By *descendants* of a directory, I mean the children of a directory, all of their children, and all of their children’s children, and so on. Directories may have descendants, but files may not. Descendants of a directory include both files and directories.

By *parent* of a file or directory, I mean the directory that contains the file or directory. Both files and directories may have a parent. Directories may be parents, but files may not.

By *ancestors* of a directory, I mean the parent of a directory, the parent’s parent, and so on. Both files and directories may have ancestors.

By *sibling* of a file or directory, I mean a file or directory that has the same parent. Both files and directories may have siblings. Files and directories may have siblings that are both files and directories.

By *disk volume* I mean a single drive letter as known to Windows. In typical home use, disk volumes are always single hard drives. (*ddeedduupp* requires that all files to be analyzed be on the same disk volume because it moves rather than copies to eliminate duplicates.)

As mentioned later in this document, *ddeedduupp* requires that files to be operated on be placed in a single directory and its descendants (named *tgt*). When the terms *parent* and *ancestor* are used, it is implicitly intended that only parents and ancestors within *tgt* and are intended.

*ddeedduupp* uses the SHA512 cryptographic hash of a file and the file’s size as reported by the operating system to determine whether two files are [almost certainly] identical. The probability that two files are different but have the same SHA512 cryptographic hash and file size is negligibly small[[1]](#footnote-1); but nonetheless I should note that for brevity I use *identical* or *duplicate* or *instance of* to mean that the files have the same size and SHA512 cryptographic hash and so are assumed to be identical. It would in principle be possible for *ddeedduupp* to mistakenly discard a unique because it mistakenly believes it is a duplicate of another file, but this is *very* unlikely.

I used *directory duplicate files* or simply *directory duplicates* to denote duplicate files that exist within the same directory. When performing directory-oriented duplication removal operations, *ddeedduupp* uses the rule that among directory duplicates, the duplicate that appears first in alphabetical sort order is retained.

## Detailed Description of *ddeedduupp* Functionality

*ddeedduupp* has the following behavior and provides the following functionality:

* Before invocation of *ddeedduupp*, a directory must be staged with exactly two children:
	+ A subdirectory named *tgt*, containing the content to be analyzed for duplicate files. This directory may contain an arbitrary number of descendants.
	+ A subdirectory named *dup*. When *ddeedduupp* removes duplicates, it places them in the *dup* directory with exactly the same relative path as they had within the *tgt* directory. The purpose of the *dup* subdirectory is to help guard against mistakes: *ddeedduupp* will only move files into the *dup* subdirectory; the user of the program has to take the final action to delete the files. Two elements of the program’s behavior should be noted:
		- *ddeedduupp* does not consider the contents of *dup* when offering deletion options to the user: the program will not delete the final instance of a file from *tgt*, even if *dup* also contains one or more instances of the file.
		- *ddeedduupp* will never under any circumstances offer the user the option of deleting the last instance of a file from *tgt*.
* *tgt* and *dup* and their descendants must be part of the same disk volume. (For the sake of speed, *ddeedduupp* moves rather than copies files from *tgt* to *dup*, and the move operation can only be performed within the same disk volume.)
* *ddeedduupp* must be invoked from the directory containing *tgt* and *dup* (i.e. at the time *ddeedduup* is invoked, the current working directory must be this directory). (There are a variety of ways to invoke *ddeedduupp* so that this constraint is met. The most common is to run *ddeedduupp* from a command prompt where the working directory is the directory containing *tgt* and *dup*.)
* Upon invocation, *ddeedduupp*:
	+ Verifies that *tgt* and *dup* exist and are the only directories.
	+ Scans the *tgt* directory and all descendants, and for each descendant file obtains the file size and calculates the SHA512 cryptographic hash.
	+ Presents the user with information about the total number of duplicates that exist within *tgt* and the approximate space that would be saved by eliminating them.
	+ Allows the user the option of navigating into directories in *tgt*. When a user navigates into a directory, information about duplicates in the directory is displayed.
	+ As the user navigates into directories, the user is provided with options for the treatment of files within the directory that are duplicates (of other files within *tgt*), and for directories that contain at least one duplicate file. (The user is provided with no options for files that are not duplicates or for directories that do not contain duplicates.)
	+ For files that are duplicates, the user is presented with the following options:
		- To move the file to *dup*.
		- To set the file as *globally authoritative*. All duplicate files within *tgt* that are not directory duplicates will be moved to *dup*.
		- To set the file as *locally authoritative*. All directory duplicates will be moved to *dup*.
		- To set the file as *ancestrally authoritative*. All duplicate descendants will be moved to *dup*.
		- To set the file as *sibling authoritative*. All duplicates within the siblings of the directory containing the file are moved to *dup*.
		- To set the file as *sibling ancestrally authoritative*. All duplicates within the siblings of the directory containing the file, as well as within the descendants of those siblings, are moved to *dup*.
	+ For directories containing directory duplicates or with siblings that contain directory duplicates, *ddeedduupp* offers the following options:
		- *Local removal*: resolve directory duplicates only within the directory being manipulated.
		- *Local and descendant removal*: resolve directory duplicates in the directory being manipulated and all its descendants. (*Note:* if this option is chosen when manipulating the *tgt* directory, the effect is global directory duplicate removal.)
		- *Sibling removal*: resolve directory duplicates in the siblings of the directory being manipulated.
		- *Sibling and descendant removal:* resolve directory duplicates in the siblings of the directory being manipulated, and in the descendants of those siblings.
	+ For directories containing duplicates that are not directory duplicates, *ddeedduupp* offers the following options:
		- *A*ll duplicates in the directory are moved to *dup*.
		- To set the directory as *globally authoritative*. All files outside the directory that are duplicates of any of those in the directory are removed.
		- To set the directory as *ancestrally authoritative*. All files in the descendants of the directory that are duplicates of those in the directory are removed.
		- To set the directory as sibling authoritative. All files in the siblings of the directory that are duplicates of those in the directory are removed.
		- To set the directory as *sibling ancestrally authoritative*. All files in the siblings of the directory containing the file that are duplicates of the files in the directory are removed, as well as duplicates within the descendants of those siblings.

# Invocation Example

# Technical Description and Internal Operation of *ddeedduupp*

## Heap Allocation

## RAM Data Structures

## Approximate Limits on File Set

## Approximate Initialization Time

## Method of Moving Files from *tgt* to *dup*

## Use on a Changing File Set

## Use on a Networked Drive

## Redistributing *ddeedduupp*

## Rebuilding *ddeedduupp* from Source Code

## Modifying *ddeedduupp*

## Redistributing modified *ddeedduupp*

1. “Negligably small” is an understatement. Paraphrasing the Wikipedia entry for *birthday attack*, in a group of 1.6 x 1068 unique files, the probability that any two would have the same SHA512 hash is 10-18. To the best of my knowledge, no collision has *ever* been found with the SHA512 hash. Still, by the pigeonhole principle, it can be proved that any file with a size of 65 bytes or greater must have at least one false duplicate file that is different but has the same length and SHA512 cryptographic hash. [↑](#footnote-ref-1)